Administrivia

- Exams will be graded over the weekend
- HW5??

Simply typed lambda-calculus with booleans

```
true : Bool  (T-TRUE)
false : Bool  (T-FALSE)
t_1 : Bool  t_2 : T  t_3 : T  (T-IF)
    if t_1 then t_2 else t_3 : T
x : T \in \Gamma  (T-VAR)
    \Gamma \vdash x : T
\Gamma, x : T_1 \vdash t_2 : T_2  (T-ABS)
    \Gamma \vdash \lambda x : T_1 . t_2 : T_1 \rightarrow T_2
\Gamma \vdash t_1 : T_1 \rightarrow T_{12}  \Gamma \vdash t_2 : T_{11}  (T-APP)
    \Gamma \vdash t_1 t_2 : T_{12}
```
**Intro vs. elim forms**

An *introduction form* for a given type gives us a way of constructing elements of this type.

An *elimination form* for a type gives us a way of using elements of this type.

---

**The Curry-Howard Correspondence**

In constructive logics, a proof of $P$ must provide evidence for $P$.

✧ "law of the excluded middle" — $P \lor \neg P$ — not recognized.

A proof of $P \land Q$ is a pair of evidence for $P$ and evidence for $Q$.

A proof of $P \supset Q$ is a procedure for transforming evidence for $P$ into evidence for $Q$.

---

**Propositions as Types**

<table>
<thead>
<tr>
<th>LOGIC</th>
<th>PROGRAMMING LANGUAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions</td>
<td>types</td>
</tr>
<tr>
<td>proposition $P \supset Q$</td>
<td>type $P \rightarrow Q$</td>
</tr>
<tr>
<td>proposition $P \land Q$</td>
<td>type $P \times Q$</td>
</tr>
<tr>
<td>proof of proposition $P$</td>
<td>term $t$ of type $P$</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
</tbody>
</table>

---

**Propositions as Types**

<table>
<thead>
<tr>
<th>LOGIC</th>
<th>PROGRAMMING LANGUAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions</td>
<td>types</td>
</tr>
<tr>
<td>proposition $P \supset Q$</td>
<td>type $P \rightarrow Q$</td>
</tr>
<tr>
<td>proposition $P \land Q$</td>
<td>type $P \times Q$</td>
</tr>
<tr>
<td>proof of proposition $P$</td>
<td>term $t$ of type $P$</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
</tbody>
</table>

---
Propositions as Types

<table>
<thead>
<tr>
<th>LOGIC</th>
<th>PROGRAMMING LANGUAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>propositions</td>
<td>types</td>
</tr>
<tr>
<td>proposition $P \supset Q$</td>
<td>type $P \rightarrow Q$</td>
</tr>
<tr>
<td>proposition $P \land Q$</td>
<td>type $P \times Q$</td>
</tr>
<tr>
<td>proof of proposition $P$</td>
<td>term $t$ of type $P$</td>
</tr>
<tr>
<td>proposition $P$ is provable</td>
<td>type $P$ is inhabited (by some term)</td>
</tr>
<tr>
<td>proof simplification</td>
<td>evaluation</td>
</tr>
<tr>
<td>(cut elimination)</td>
<td></td>
</tr>
</tbody>
</table>

Erasure

\[
\begin{align*}
\text{erase}(x) & \rightarrow x \\
\text{erase}(\lambda x: T_1. t_2) & \rightarrow \lambda x. \text{erase}(t_2) \\
\text{erase}(t_1, t_2) & \rightarrow \text{erase}(t_1) \text{ erase}(t_2)
\end{align*}
\]

Typability

An untyped $\lambda$-term $m$ is said to be **typable** if there is some term $t$ in the simply typed lambda-calculus, some type $T$, and some context $\Gamma$ such that $\text{erase}(t) = m$ and $\Gamma \vdash t : T$.

Cf. type reconstruction in OCaml.

On to real programming languages...
Base types

Up to now, we've formulated "base types" (e.g. \(\text{Nat}\)) by adding them to the syntax of types, extending the syntax of terms with associated constants (\(\text{zero}\)) and operators (\(\text{succ}\), etc.) and adding appropriate typing and evaluation rules. We can do this for as many base types as we like.

For more theoretical discussions (as opposed to programming) we can often ignore the term-level inhabitants of base types, and just treat these types as uninterpreted constants.

E.g., suppose \(B\) and \(C\) are some base types. Then we can ask (without knowing anything more about \(B\) or \(C\)) whether there are any types \(S\) and \(T\) such that the term
\[
(\lambda z : S.\ \lambda g : T.\ f\ g)\ (\lambda x : B.\ z)
\]
is well typed.

The Unit type

\[
t ::= \ldots
\]
\[
\text{unit}
\]
\[
\text{terms}
\]
\[
v ::= \ldots
\]
\[
\text{values}
\]
\[
\text{constant unit}
\]
\[
T ::= \ldots
\]
\[
\text{unit type}
\]
\[
\text{types}
\]
\[
\Gamma \vdash \text{unit : Unit}
\]

New typing rules

\[
\Gamma \vdash t : T
\]

Sequencing

\[
t ::= \ldots
\]
\[
t_1 ; t_2
\]

Sequencing

\[
t ::= \ldots
\]
\[
t_1 ; t_2
\]
\[
t_1 \rightarrow t'_1
\]
\[
t_1 ; t_2 \rightarrow t'_1 ; t_2
\]

\[
\text{(E-SEQ)}
\]
\[
\text{unit} ; t_2 \rightarrow t_2
\]

\[
\text{(E-SEQNEXT)}
\]
\[
\Gamma \vdash t_1 : \text{Unit}\quad \Gamma \vdash t_2 : T_2
\]
\[
\Gamma \vdash t_1 ; t_2 : T_2
\]

\[
\text{(T-SEQ)}
\]
Derived forms

- Syntactic sugar
- Internal language vs. external (surface) language

Sequencing as a derived form

\[ t_1; t_2 \overset{\text{def}}{=} (\lambda x: \text{Unit}. t_2) \, t_1 \]
where \( x \not\in \text{FV}(t_2) \)

Equivalence of the two definitions

New syntactic forms

\[ t ::= \ldots \]
\[ t \text{ as } T \]

New evaluation rules

\[ v_1 \text{ as } T \rightarrow v_1 \]
\[ t_1 \rightarrow t'_1 \]
\[ t_1 \text{ as } T \rightarrow t'_1 \text{ as } T \]

New typing rules

\[ \Gamma \vdash t_1 : T \]
\[ \Gamma \vdash t_1 \text{ as } T : T \]

Ascription

Newsyntactic forms

\[ \text{::=} \ldots \]
\[ \text{terms ascription} \]

New evaluation rules

\[ (E\text{-ASCRIBE}) \]
\[ (E\text{-ASCRIBE1}) \]

New typing rules

\[ \Gamma \vdash t_1 : T \]
\[ (T\text{-ASCRIBE}) \]
Ascription as a derived form

t as T \overset{\text{def}}{=} (\lambda x : T. \ x) \ t

Let-bindings

New syntactic forms
\[ t ::= \ldots \]
\[ \text{let } x = t \text{ in } t \]
\[ \text{let binding } \]

New evaluation rules
\[ \text{let } x = v_1 \text{ in } t_2 \rightarrow [x \mapsto v_1]_{t_2} \]
\[ t_1 \rightarrow t'_1 \]
\[ \text{let } x = t_1 \text{ in } t_2 \rightarrow \text{let } x = t'_1 \text{ in } t_2 \]

New typing rules
\[ \Gamma \vdash t : T \]
\[ \Gamma , x : T_1 \vdash t_2 : T_2 \]
\[ \Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : T_2 \]
(E-LET)

Pairs

\[ t ::= \ldots \]
\[ \{t, t\} \]
\[ t.1 \]
\[ t.2 \]

\[ v ::= \ldots \]
\[ \{v, v\} \]

\[ T ::= \ldots \]
\[ T_1 \times T_2 \]

Evaluation rules for pairs
\[ \{v_1, v_2\}.1 \rightarrow v_1 \]
\[ \{v_1, v_2\}.2 \rightarrow v_2 \]
\[ t_1 \rightarrow t'_1 \]
\[ t_1.1 \rightarrow t'_1.1 \]
\[ t_1.2 \rightarrow t'_1.2 \]
\[ t_1 \rightarrow t'_1 \]
\[ \{t_1, t_2\} \rightarrow \{t'_1, t_2\} \]
\[ t_2 \rightarrow t'_2 \]
\[ \{v_1, v_2\} \rightarrow \{v'_1, v_2\} \]
### Typing rules for pairs

\[
\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2 \\
\frac{}{\Gamma \vdash \langle t_1, t_2 \rangle : T_1 \times T_2} \quad \text{(T-PAIR)}
\]

\[
\Gamma \vdash t_1 : T_{11} \times T_{12} \\
\frac{}{\Gamma \vdash t_1.1 : T_{11}} \quad \text{(T-PROJ1)}
\]

\[
\Gamma \vdash t_1 : T_{11} \times T_{12} \\
\frac{}{\Gamma \vdash t_1.2 : T_{12}} \quad \text{(T-PROJ2)}
\]

### Tuples

\[
t ::= \ldots \\
\{ t_i \}_{i \in \mathbb{N}} \\
t_i
\]

\[
v ::= \ldots \\
\{ t_i \}_{i \in \mathbb{N}} \\
tuplevalue
\]

\[
T ::= \ldots \\
\{ T_i \}_{i \in \mathbb{N}} \\
tupletype
\]

### Evaluation rules for tuples

\[
\{ t_i \}_{i \in \mathbb{N}}, j \rightarrow t_j \quad \text{(E-PROJ)}
\]

\[
\frac{t \rightarrow t'}{t_1 \rightarrow t'_{1.1}} \quad \text{(E-PROJ)}
\]

\[
\frac{t \rightarrow t'}{\{ t_1 \}_{i \in \mathbb{N}}, \{ t_i \}_{i \in \mathbb{N}}} \rightarrow \{ t_1 \}_{i \in \mathbb{N}}, \{ t_i' \}_{i \in \mathbb{N}} \quad \text{(E-TUPLE)}
\]

### Typing rules for tuples

\[
\text{for each } i \quad \Gamma \vdash t_i : T_i \\
\frac{}{\Gamma \vdash \{ t_i \}_{i \in \mathbb{N}} : \{ T_i \}_{i \in \mathbb{N}}} \quad \text{(T-TUPLE)}
\]

\[
\frac{}{\Gamma \vdash t_1 : \{ T_i \}_{i \in \mathbb{N}}} \\
\frac{}{\Gamma \vdash t_1.1 : T_i} \quad \text{(T-PROJ)}
\]
### Records

\[
\begin{align*}
\text{t} & ::= \ldots \quad \text{terms} \\
\{l_i = t_i \mid i \in \{1, \ldots, n\}\} & \quad \text{record} \\
\text{t}.l & \quad \text{projection} \\
\text{v} & ::= \ldots \quad \text{values} \\
\{l_i = v_i \mid i \in \{1, \ldots, n\}\} & \quad \text{record value} \\
\text{T} & ::= \ldots \quad \text{types} \\
\{l_i : T_i \mid i \in \{1, \ldots, n\}\} & \quad \text{type of records}
\end{align*}
\]

### Evaluation rules for records

\[
\begin{align*}
\{l_i = v_i \mid i \in \{1, \ldots, n\}\}, l_i & \to v_i & (E-\text{PROJRcd}) \\
\frac{}{t_i \to t'_i} & (E-\text{PROJ}) \\
\frac{}{t_i.l \to t'_i.l} & (E-\text{PROJ}) \\
\frac{}{t_i \to t'_i} & (E-\text{Rcd}) \\
\frac{}{\{l_i = v_i \mid i \in \{1, \ldots, n\}, l_i = t_i, l_k = t_k \mid k \in \{1, \ldots, n\}\} \to \{l_i = v_i \mid i \in \{1, \ldots, n\}, l_i = t'_i, l_k = t_k \mid k \in \{1, \ldots, n\}\}} & (E-\text{Rcd})
\end{align*}
\]

### Typing rules for records

\[
\begin{align*}
\text{for each } i & \quad \Gamma \vdash t_i : T_i & (T-\text{Rcd}) \\
\frac{}{\Gamma \vdash \{l_i = t_i \mid i \in \{1, \ldots, n\}\}} & (T-\text{Rcd}) \\
\frac{}{\Gamma \vdash \{l_i : T_i \mid i \in \{1, \ldots, n\}\}} & (T-\text{PROJ}) \\
\frac{}{\Gamma \vdash t_i.l : T_i} & (T-\text{PROJ})
\end{align*}
\]